Concrete Semantics

with Isabelle/HOL

Peter Lammich (borrowed from Tobias Nipkow)

ECE
Virginia Tech

2017-11-13

Part II

Semantics

Chapter 7

IMP:
A Simple Imperative Language

1. IMP Commands

2. Big-Step Semantics

3. Small-Step Semantics

1. IMP Commands
2. Big-Step Semantics
3. Small-Step Semantics

Terminology

Statement: declaration of fact or claim

Terminology

Statement: declaration of fact or claim

Semantics is easy.

Terminology

Statement: declaration of fact or claim

Semantics is easy.

Command: order to do something

Terminology

Statement: declaration of fact or claim

Semantics is easy.

Command: order to do something

Study the book until you have understood it.

Terminology

Statement: declaration of fact or claim

Semantics is easy.

Command: order to do something

Study the book until you have understood it.

Expressions are *evaluated*, commands are *executed*

Commands

Concrete syntax:

$$
\begin{array}{rcl}
com & ::= & \texttt{SKIP} \\
& | & string\ \texttt{::=}\ aexp \\
& | & com\ \texttt{;;}\ com \\
& | & \texttt{IF}\ bexp\ \texttt{THEN}\ com\ \texttt{ELSE}\ com \\
& | & \texttt{WHILE}\ bexp\ \texttt{DO}\ com
\end{array}
$$

Commands

Abstract syntax:

$$
\begin{array}{lcl}
\textbf{datatype}\ com & = & SKIP \\
& | & Assign\ string\ aexp \\
& | & Seq\ com\ com \\
& | & If\ \ bexp\ com\ com \\
& | & While\ bexp\ com
\end{array}
$$

Com.thy

Big-step semantics

Concrete syntax:

$$(com,\ initial\text{-}state) \Rightarrow final\text{-}state$$

Big-step semantics

Concrete syntax:

$$(com,\ initial\text{-}state) \Rightarrow final\text{-}state$$

Intended meaning of $(c,\ s) \Rightarrow t$:

Big-step semantics

Concrete syntax:

$$(com,\ initial\text{-}state) \Rightarrow final\text{-}state$$

Intended meaning of $(c,\ s) \Rightarrow t$:

Command c started in state s terminates in state t

Big-step semantics

Concrete syntax:

$$(com,\ initial\text{-}state) \Rightarrow final\text{-}state$$

Intended meaning of $(c,\ s) \Rightarrow t$:

Command c started in state s terminates in state t

"$\Rightarrow$" here not type!

Big-step rules

$$(SKIP,\ s) \Rightarrow\ s$$

Big-step rules

$$(SKIP,\ s) \Rightarrow\ s$$

$$(x ::=\ a,\ s) \Rightarrow\ s(x :=\ aval\ a\ s)$$

Big-step rules

$$(SKIP,\ s) \Rightarrow s$$

$$(x ::= a,\ s) \Rightarrow s(x := aval\ a\ s)$$

$$\frac{(c_1,\ s_1) \Rightarrow s_2 \qquad (c_2,\ s_2) \Rightarrow s_3}{(c_1;;\ c_2,\ s_1) \Rightarrow s_3}$$

Big-step rules

$$\frac{bval\ b\ s \qquad (c_1,\ s) \Rightarrow t}{(IF\ b\ THEN\ c_1\ ELSE\ c_2,\ s) \Rightarrow t}$$

Big-step rules

$$\frac{bval\ b\ s \qquad (c_1,\ s) \Rightarrow t}{(IF\ b\ THEN\ c_1\ ELSE\ c_2,\ s) \Rightarrow t}$$

$$\frac{\neg\ bval\ b\ s \qquad (c_2,\ s) \Rightarrow t}{(IF\ b\ THEN\ c_1\ ELSE\ c_2,\ s) \Rightarrow t}$$

Big-step rules

$$\frac{\neg\ bval\ b\ s}{(WHILE\ b\ DO\ c,\ s) \Rightarrow s}$$

Big-step rules

$$\frac{\neg\ bval\ b\ s}{(WHILE\ b\ DO\ c,\ s) \Rightarrow s}$$

$$\frac{bval\ b\ s_1 \quad (c,\ s_1) \Rightarrow s_2 \quad (WHILE\ b\ DO\ c,\ s_2) \Rightarrow s_3}{(WHILE\ b\ DO\ c,\ s_1) \Rightarrow s_3}$$

Logically speaking

$$(c,\ s) \Rightarrow\ t$$

is just infix syntax for

$$big\_step\ (c{,}s)\ \ t$$

Logically speaking

$$(c,\ s) \Rightarrow t$$

is just infix syntax for

$$big\_step\ (c,s)\ t$$

where

$$big\_step :: com \times state \Rightarrow state \Rightarrow bool$$

is an inductively defined predicate.

Big_Step.thy

Semantics

Rule inversion

What can we deduce from

- $(SKIP, s) \Rightarrow t$?

Rule inversion

What can we deduce from

- $(SKIP,\ s) \Rightarrow t$? $\qquad t = s$

Rule inversion

What can we deduce from

- $(SKIP,\ s) \Rightarrow t$? $t = s$
- $(x ::= a,\ s) \Rightarrow t$?

Rule inversion

What can we deduce from

- $(SKIP,\ s) \Rightarrow t$? $\quad t = s$
- $(x ::= a,\ s) \Rightarrow t$? $\quad t = s(x := aval\ a\ s)$

Rule inversion

What can we deduce from

- $(SKIP,\ s) \Rightarrow t$? $\quad t = s$
- $(x ::= a,\ s) \Rightarrow t$? $\quad t = s(x := aval\ a\ s)$
- $(c_1;;\ c_2,\ s_1) \Rightarrow s_3$?

Rule inversion

What can we deduce from

- $(SKIP,\ s) \Rightarrow t$? $\quad t = s$
- $(x ::= a,\ s) \Rightarrow t$? $\quad t = s(x := aval\ a\ s)$
- $(c_1;;\ c_2,\ s_1) \Rightarrow s_3$?
  $\exists\, s_2.\ (c_1,\ s_1) \Rightarrow s_2 \wedge (c_2,\ s_2) \Rightarrow s_3$

Rule inversion

What can we deduce from

- $(SKIP,\ s) \Rightarrow t$? $\quad t = s$
- $(x ::= a,\ s) \Rightarrow t$? $\quad t = s(x := aval\ a\ s)$
- $(c_1;;\ c_2,\ s_1) \Rightarrow s_3$?
  $\exists\, s_2.\ (c_1,\ s_1) \Rightarrow s_2 \wedge (c_2,\ s_2) \Rightarrow s_3$
- $(IF\ b\ THEN\ c_1\ ELSE\ c_2,\ s) \Rightarrow t$?

Rule inversion

What can we deduce from

- $(SKIP,\ s) \Rightarrow t$? $\quad t = s$
- $(x ::= a,\ s) \Rightarrow t$? $\quad t = s(x := aval\ a\ s)$
- $(c_1;;\ c_2,\ s_1) \Rightarrow s_3$?
  $\exists\, s_2.\ (c_1,\ s_1) \Rightarrow s_2 \wedge (c_2,\ s_2) \Rightarrow s_3$
- $(IF\ b\ THEN\ c_1\ ELSE\ c_2,\ s) \Rightarrow t$?
  $bval\ b\ s \wedge (c_1,\ s) \Rightarrow t\ \vee$
  $\neg\ bval\ b\ s \wedge (c_2,\ s) \Rightarrow t$

Rule inversion

What can we deduce from

- $(SKIP,\ s) \Rightarrow t$? $\quad t = s$
- $(x ::= a,\ s) \Rightarrow t$? $\quad t = s(x := aval\ a\ s)$
- $(c_1;;\ c_2,\ s_1) \Rightarrow s_3$?
  $\exists\, s_2.\ (c_1,\ s_1) \Rightarrow s_2 \wedge (c_2,\ s_2) \Rightarrow s_3$
- $(IF\ b\ THEN\ c_1\ ELSE\ c_2,\ s) \Rightarrow t$?
  $bval\ b\ s \wedge (c_1,\ s) \Rightarrow t\ \vee$
  $\neg\ bval\ b\ s \wedge (c_2,\ s) \Rightarrow t$
- $(w,\ s) \Rightarrow t$ where $w = WHILE\ b\ DO\ c$?

Rule inversion

What can we deduce from

- $(SKIP,\ s) \Rightarrow t$? $\quad t = s$
- $(x ::= a,\ s) \Rightarrow t$? $\quad t = s(x := aval\ a\ s)$
- $(c_1;;\ c_2,\ s_1) \Rightarrow s_3$?
  $\exists\, s_2.\ (c_1,\ s_1) \Rightarrow s_2 \wedge (c_2,\ s_2) \Rightarrow s_3$
- $(IF\ b\ THEN\ c_1\ ELSE\ c_2,\ s) \Rightarrow t$?
  $bval\ b\ s \wedge (c_1,\ s) \Rightarrow t\ \vee$
  $\neg\ bval\ b\ s \wedge (c_2,\ s) \Rightarrow t$
- $(w,\ s) \Rightarrow t$ where $w = WHILE\ b\ DO\ c$?
  $\neg\ bval\ b\ s \wedge t = s\ \vee$
  $bval\ b\ s \wedge (\exists\, s'.\ (c,\ s) \Rightarrow s' \wedge (w,\ s') \Rightarrow t)$

Automating rule inversion

Isabelle command **inductive_cases** produces theorems that perform rule inversions automatically.

We reformulate the inverted rules. Example:

$$\frac{(c_1;;\ c_2,\ s_1) \Rightarrow\ s_3}{\exists\, s_2.\ (c_1,\ s_1) \Rightarrow\ s_2 \wedge\ (c_2,\ s_2) \Rightarrow\ s_3}$$

We reformulate the inverted rules. Example:

$$\frac{(c_1;;\ c_2,\ s_1) \Rightarrow s_3}{\exists\, s_2.\ (c_1,\ s_1) \Rightarrow s_2 \wedge (c_2,\ s_2) \Rightarrow s_3}$$

is logically equivalent to

$$\frac{(c_1;;\ c_2,\ s_1) \Rightarrow s_3 \qquad \bigwedge s_2.\ [\![(c_1,\ s_1) \Rightarrow s_2;\ (c_2,\ s_2) \Rightarrow s_3]\!] \Longrightarrow P}{P}$$

We reformulate the inverted rules. Example:

$$\frac{(c_1;;\ c_2,\ s_1) \Rightarrow s_3}{\exists\, s_2.\ (c_1,\ s_1) \Rightarrow s_2 \wedge (c_2,\ s_2) \Rightarrow s_3}$$

is logically equivalent to

$$\frac{(c_1;;\ c_2,\ s_1) \Rightarrow s_3 \qquad \bigwedge s_2.\ [\![(c_1,\ s_1) \Rightarrow s_2;\ (c_2,\ s_2) \Rightarrow s_3]\!] \Longrightarrow P}{P}$$

Replaces assm $(c_1;;\ c_2,\ s_1) \Rightarrow s_3$ by two assms $(c_1,\ s_1) \Rightarrow s_2$ and $(c_2,\ s_2) \Rightarrow s_3$

We reformulate the inverted rules. Example:

$$\frac{(c_1;;\ c_2,\ s_1) \Rightarrow s_3}{\exists\, s_2.\ (c_1,\ s_1) \Rightarrow s_2 \wedge (c_2,\ s_2) \Rightarrow s_3}$$

is logically equivalent to

$$\frac{(c_1;;\ c_2,\ s_1) \Rightarrow s_3 \qquad \bigwedge s_2.\ [\![(c_1,\ s_1) \Rightarrow s_2;\ (c_2,\ s_2) \Rightarrow s_3]\!] \Longrightarrow P}{P}$$

Replaces assm $(c_1;;\ c_2,\ s_1) \Rightarrow s_3$ by two assms $(c_1,\ s_1) \Rightarrow s_2$ and $(c_2,\ s_2) \Rightarrow s_3$ (with a new fixed s_2).

We reformulate the inverted rules. Example:

$$\frac{(c_1;;\ c_2,\ s_1) \Rightarrow s_3}{\exists\, s_2.\ (c_1,\ s_1) \Rightarrow s_2 \wedge (c_2,\ s_2) \Rightarrow s_3}$$

is logically equivalent to

$$\frac{(c_1;;\ c_2,\ s_1) \Rightarrow s_3 \qquad \bigwedge s_2.\ [\![(c_1,\ s_1) \Rightarrow s_2;\ (c_2,\ s_2) \Rightarrow s_3]\!] \Longrightarrow P}{P}$$

Replaces assm $(c_1;;\ c_2,\ s_1) \Rightarrow s_3$ by two assms $(c_1,\ s_1) \Rightarrow s_2$ and $(c_2,\ s_2) \Rightarrow s_3$ (with a new fixed s_2).
No $\exists$ and $\wedge$!

The general format: *elimination rules*

$$\frac{asm \quad asm_1 \Longrightarrow P \quad \dots \quad asm_n \Longrightarrow P}{P}$$

The general format: *elimination rules*

$$\frac{asm \quad asm_1 \Longrightarrow P \quad \ldots \quad asm_n \Longrightarrow P}{P}$$

(possibly with $\bigwedge \overline{x}$ in front of the $asm_i \Longrightarrow P$)

The general format: *elimination rules*

$$\frac{asm \quad asm_1 \Longrightarrow P \quad \ldots \quad asm_n \Longrightarrow P}{P}$$

(possibly with $\bigwedge \overline{x}$ in front of the $asm_i \Longrightarrow P$)

Reading:

To prove a goal P with assumption asm,
prove all $asm_i \Longrightarrow P$

The general format: *elimination rules*

$$\frac{asm \quad asm_1 \Longrightarrow P \quad \ldots \quad asm_n \Longrightarrow P}{P}$$

(possibly with $\bigwedge \overline{x}$ in front of the $asm_i \Longrightarrow P$)

Reading:

To prove a goal P with assumption asm,
prove all $asm_i \Longrightarrow P$

Example:

$$\frac{F \ \vee \ G \quad F \ \Longrightarrow \ P \quad G \ \Longrightarrow \ P}{P}$$

elim attribute

- Theorems with *elim* attribute are used automatically by *blast*, *fastforce* and *auto*

elim attribute

- Theorems with *elim* attribute are used automatically by *blast*, *fastforce* and *auto*
- Can also be added locally, eg *(blast elim: ...)*

elim attribute

- Theorems with *elim* attribute are used automatically by *blast*, *fastforce* and *auto*
- Can also be added locally, eg *(blast elim: ...)*
- Variant: *elim!* applies elim-rules eagerly.

Big_Step.thy

Rule inversion

Execution is deterministic

Any two executions of the same command in the same start state lead to the same final state:

$$(c,\ s) \Rightarrow t \implies (c,\ s) \Rightarrow t' \implies t = t'$$

Execution is deterministic

Any two executions of the same command in the same start state lead to the same final state:

$$(c,\ s) \Rightarrow t \implies (c,\ s) \Rightarrow t' \implies t = t'$$

Proof by rule induction, for arbitrary t'.

`Big_Step.thy`

Execution is deterministic

The boon and bane of big steps

We cannot observe intermediate states/steps

The boon and bane of big steps

We cannot observe intermediate states/steps

Example problem:

The boon and bane of big steps

We cannot observe intermediate states/steps

Example problem:

(c,s) does not terminate iff $\nexists\, t.\ (c,\ s) \Rightarrow\ t$?

The boon and bane of big steps

We cannot observe intermediate states/steps

Example problem:

$$(c,s) \text{ does not terminate iff } \nexists\, t.\ (c,\ s) \Rightarrow\ t\ ?$$

Needs a formal notion of nontermination to prove it.

The boon and bane of big steps

We cannot observe intermediate states/steps

Example problem:

$$(c,s) \text{ does not terminate iff } \nexists\, t.\ (c,\ s) \Rightarrow\ t\ ?$$

Needs a formal notion of nontermination to prove it.
Could be wrong if we have forgotten a $\Rightarrow$ rule.

Big-step semantics cannot directly describe

- nonterminating computations,

Big-step semantics cannot directly describe

- nonterminating computations,
- parallel computations.

Big-step semantics cannot directly describe

- nonterminating computations,
- parallel computations.

We need a finer grained semantics!

1. IMP Commands

2. Big-Step Semantics

3. Small-Step Semantics

Small-step semantics

Concrete syntax:

$$(com, state) \to (com, state)$$

Small-step semantics

Concrete syntax:

$$(com, state) \to (com, state)$$

Intended meaning of $(c,\ s) \to (c',\ s')$:

Small-step semantics

Concrete syntax:

$$(com, state) \to (com, state)$$

Intended meaning of $(c,\ s) \to (c',\ s')$:

> *The first step in the execution of c in state s*
> *leaves a "remainder" command c'*
> *to be executed in state s'.*

Small-step semantics

Concrete syntax:

$$(com, state) \rightarrow (com, state)$$

Intended meaning of $(c,\ s) \rightarrow (c',\ s')$:

> *The first step in the execution of* c *in state* s
> *leaves a "remainder" command* c'
> *to be executed in state* s'*.*

Execution as finite or infinite reduction:

$$(c_1, s_1) \rightarrow (c_2, s_2) \rightarrow (c_3, s_3) \rightarrow \ldots$$

Terminology

- A pair (c, s) is called a *configuration*.

Terminology

- A pair (c,s) is called a *configuration*.
- If $cs \to cs'$ we say that cs *reduces* to cs'.

Terminology

- A pair (c,s) is called a *configuration*.
- If $cs \to cs'$ we say that cs *reduces* to cs'.
- A configuration cs is *final* iff $\nexists\, cs'.\ cs \to cs'$

The intention:

$$(SKIP,\ s) \ \ \text{is final}$$

The intention:

$$(SKIP,\ s) \ \text{ is final}$$

Why?

$SKIP$ is the empty program.

The intention:

$$(SKIP,\ s) \ \text{ is final}$$

Why?

$SKIP$ is the empty program. Nothing more to be done.

Small-step rules

$$(x ::= a,\ s) \ \rightarrow$$

Small-step rules

$$(x ::= a,\ s) \ \rightarrow \ (SKIP,\ s(x := \ aval\ a\ s))$$

Small-step rules

$$(x ::= a,\ s) \ \rightarrow\ (SKIP,\ s(x := \ aval\ a\ s))$$

$$(SKIP;;\ c,\ s) \ \rightarrow$$

Small-step rules

$$(x ::= a,\ s) \ \rightarrow\ (SKIP,\ s(x := \ aval\ a\ s))$$

$$(SKIP;;\ c,\ s) \ \rightarrow\ (c,\ s)$$

Small-step rules

$$(x ::= a,\ s) \ \rightarrow\ (SKIP,\ s(x := \ aval\ a\ s))$$

$$(SKIP;;\ c,\ s) \ \rightarrow\ (c,\ s)$$

$$\frac{(c_1, s) \ \rightarrow\ (c_1', s')}{(c_1;; c_2, s) \ \rightarrow}$$

Small-step rules

$$(x ::= a,\ s) \ \rightarrow \ (SKIP,\ s(x := \ aval \ a \ s))$$

$$(SKIP;;\ c,\ s) \ \rightarrow \ (c,\ s)$$

$$\frac{(c_1, s) \ \rightarrow \ (c_1', s')}{(c_1;; c_2, s) \ \rightarrow \ (c_1';; c_2, s')}$$

Small-step rules

$$\frac{bval\ b\ s}{(IF\ b\ THEN\ c_1\ ELSE\ c_2, s)\ \rightarrow}$$

Small-step rules

$$\frac{bval\ b\ s}{(IF\ b\ THEN\ c_1\ ELSE\ c_2, s) \ \rightarrow\ (c_1, s)}$$

Small-step rules

$$\frac{bval\ b\ s}{(IF\ b\ THEN\ c_1\ ELSE\ c_2, s) \ \rightarrow\ (c_1, s)}$$

$$\frac{\neg\ bval\ b\ s}{(IF\ b\ THEN\ c_1\ ELSE\ c_2, s) \ \rightarrow\ (c_2, s)}$$

Small-step rules

$$\frac{bval\ b\ s}{(IF\ b\ THEN\ c_1\ ELSE\ c_2, s)\ \rightarrow\ (c_1, s)}$$

$$\frac{\neg\ bval\ b\ s}{(IF\ b\ THEN\ c_1\ ELSE\ c_2, s)\ \rightarrow\ (c_2, s)}$$

$$(WHILE\ b\ DO\ c,\ s)\ \rightarrow$$

Small-step rules

$$\frac{bval\ b\ s}{(IF\ b\ THEN\ c_1\ ELSE\ c_2, s) \ \rightarrow\ (c_1, s)}$$

$$\frac{\neg\ bval\ b\ s}{(IF\ b\ THEN\ c_1\ ELSE\ c_2, s) \ \rightarrow\ (c_2, s)}$$

$$(WHILE\ b\ DO\ c,\ s) \ \rightarrow$$
$$(IF\ b\ THEN\ c;;\ WHILE\ b\ DO\ c\ ELSE\ SKIP,\ s)$$

Small-step rules

$$\frac{bval\ b\ s}{(IF\ b\ THEN\ c_1\ ELSE\ c_2, s)\ \rightarrow\ (c_1, s)}$$

$$\frac{\neg\ bval\ b\ s}{(IF\ b\ THEN\ c_1\ ELSE\ c_2, s)\ \rightarrow\ (c_2, s)}$$

$$(WHILE\ b\ DO\ c,\ s)\ \rightarrow$$
$$(IF\ b\ THEN\ c;;\ WHILE\ b\ DO\ c\ ELSE\ SKIP,\ s)$$

Fact $(SKIP, s)$ is a final configuration.

Small-step examples

$$(''z'' ::= V\ ''x'';;\ ''x'' ::= V\ ''y'';;\ ''y'' ::= V\ ''z'',\ s) \to$$
$$\ldots$$

where $s = <''x'' := 3,\ ''y'' := 7,\ ''z'' := 5>$.

Small-step examples

$$(''z'' ::= V\ ''x'';;\ ''x'' ::= V\ ''y'';;\ ''y'' ::= V\ ''z'',\ s) \rightarrow$$

$$\ldots$$

where $s = <''x'' := 3,\ ''y'' := 7,\ ''z'' := 5>$.

$$(w,\ s_0) \rightarrow \ldots$$

where
$$\begin{array}{lcl} w & = & WHILE\ b\ DO\ c \\ b & = & Less\ (V\ ''x'')\ (N\ 1) \\ c & = & ''x'' ::= Plus\ (V\ ''x'')\ (N\ 1) \\ s_n & = & <''x'' := n> \end{array}$$

Small_Step.thy

Semantics

Are big and small-step semantics equivalent?

From $\Rightarrow$ to $\rightarrow*$

From $\Rightarrow$ to $\rightarrow*$

Theorem $cs \Rightarrow t \implies cs \rightarrow* (SKIP,\ t)$

From $\Rightarrow$ to $\rightarrow*$

Theorem $cs \Rightarrow t \implies cs \rightarrow* (SKIP, t)$

Proof by rule induction

From $\Rightarrow$ to $\rightarrow*$

Theorem $cs \Rightarrow t \implies cs \rightarrow* (SKIP, t)$

Proof by rule induction (of course on $cs \Rightarrow t$)

From $\Rightarrow$ to $\rightarrow*$

Theorem $cs \Rightarrow t \implies cs \rightarrow* (SKIP, t)$

Proof by rule induction (of course on $cs \Rightarrow t$)

In two cases a lemma is needed:

From $\Rightarrow$ to $\rightarrow*$

Theorem $cs \Rightarrow t \implies cs \rightarrow* (SKIP, t)$

Proof by rule induction (of course on $cs \Rightarrow t$)

In two cases a lemma is needed:

Lemma

$(c_1, s) \rightarrow* (c_1{}', s') \implies (c_1;; c_2, s) \rightarrow* (c_1{}';; c_2, s')$

From $\Rightarrow$ to $\rightarrow*$

Theorem $cs \Rightarrow t \implies cs \rightarrow* (SKIP,\ t)$

Proof by rule induction (of course on $cs \Rightarrow t$)
In two cases a lemma is needed:

Lemma
$(c_1,\ s) \rightarrow* ({c_1}',\ s') \implies (c_1;;\ c_2,\ s) \rightarrow* ({c_1}';;\ c_2,\ s')$

Proof by rule induction.

From $\rightarrow*$ to $\Rightarrow$

From $\rightarrow*$ to $\Rightarrow$

Theorem $cs \rightarrow* (SKIP,\ t) \implies cs \Rightarrow t$

From $\rightarrow*$ to $\Rightarrow$

Theorem $cs \rightarrow* (SKIP, t) \implies cs \Rightarrow t$

Proof by rule induction on $cs \rightarrow* (SKIP, t)$.

From $\rightarrow*$ to $\Rightarrow$

Theorem $cs \rightarrow* (SKIP,\, t) \implies cs \Rightarrow t$

Proof by rule induction on $cs \rightarrow* (SKIP,\, t)$.
In the induction step a lemma is needed:

From $\rightarrow*$ to $\Rightarrow$

Theorem $cs \rightarrow* (SKIP,\ t) \implies cs \Rightarrow t$

Proof by rule induction on $cs \rightarrow* (SKIP,\ t)$.
In the induction step a lemma is needed:

Lemma $cs \rightarrow cs' \implies cs' \Rightarrow t \implies cs \Rightarrow t$

From $\rightarrow*$ to $\Rightarrow$

Theorem $cs \rightarrow* (SKIP, t) \implies cs \Rightarrow t$

Proof by rule induction on $cs \rightarrow* (SKIP, t)$.
In the induction step a lemma is needed:

Lemma $cs \rightarrow cs' \implies cs' \Rightarrow t \implies cs \Rightarrow t$

Proof by rule induction on $cs \rightarrow cs'$.

Equivalence

Corollary $cs \Rightarrow t \;\longleftrightarrow\; cs \rightarrow* (SKIP,\ t)$

`Small_Step.thy`

Equivalence of big and small

Can execution stop prematurely?

Can execution stop prematurely?

That is, are there any final configs except $(SKIP, s)$?

Can execution stop prematurely?

That is, are there any final configs except $(SKIP, s)$?

Lemma $\mathit{final}\ (c,\ s) \Longrightarrow c = SKIP$

Can execution stop prematurely?

That is, are there any final configs except $(SKIP, s)$?

Lemma $\mathit{final}\ (c,\ s) \Longrightarrow c = SKIP$

We prove the contrapositive

$c \neq SKIP \Longrightarrow \neg\ \mathit{final}(c, s)$

by induction on c.

Can execution stop prematurely?

That is, are there any final configs except $(SKIP, s)$?

Lemma $\mathit{final}\ (c,\ s) \Longrightarrow c = SKIP$

We prove the contrapositive

$c \neq SKIP \Longrightarrow \neg\ \mathit{final}(c, s)$

by induction on c.

- Case $c_1;;\ c_2$: by case distinction:

Can execution stop prematurely?

That is, are there any final configs except $(SKIP,s)$?

Lemma $final\ (c,\ s) \Longrightarrow c = SKIP$

We prove the contrapositive

$c \neq SKIP \Longrightarrow \neg\ final(c,s)$

by induction on c.

- Case $c_1;;\ c_2$: by case distinction:
  - $c_1 = SKIP$

Can execution stop prematurely?

That is, are there any final configs except $(SKIP, s)$?

Lemma $\mathit{final}\ (c,\ s) \Longrightarrow c = SKIP$

We prove the contrapositive

$c \neq SKIP \Longrightarrow \neg\ \mathit{final}(c, s)$

by induction on c.

- Case $c_1;;\ c_2$: by case distinction:
  - $c_1 = SKIP \Longrightarrow \neg\ \mathit{final}\ (c_1;;\ c_2,\ s)$

Can execution stop prematurely?

That is, are there any final configs except $(SKIP, s)$?

Lemma $\mathit{final}\ (c,\ s) \Longrightarrow c = SKIP$

We prove the contrapositive

$c \neq SKIP \Longrightarrow \neg\ \mathit{final}(c, s)$

by induction on c.

- Case $c_1;;\ c_2$: by case distinction:
  - $c_1 = SKIP \Longrightarrow \neg\ \mathit{final}\ (c_1;;\ c_2,\ s)$
  - $c_1 \neq SKIP$

Can execution stop prematurely?

That is, are there any final configs except $(SKIP, s)$?

Lemma $\mathit{final}\ (c,\ s) \Longrightarrow c = SKIP$

We prove the contrapositive

$c \neq SKIP \Longrightarrow \neg\ \mathit{final}(c, s)$

by induction on c.

- Case $c_1;;\ c_2$: by case distinction:
  - $c_1 = SKIP \Longrightarrow \neg\ \mathit{final}\ (c_1;;\ c_2,\ s)$
  - $c_1 \neq SKIP \Longrightarrow \neg\ \mathit{final}\ (c_1,\ s)$

Can execution stop prematurely?

That is, are there any final configs except $(SKIP, s)$?

Lemma $\mathit{final}\ (c,\ s) \Longrightarrow c = SKIP$

We prove the contrapositive

$c \neq SKIP \Longrightarrow \neg\ \mathit{final}(c, s)$

by induction on c.

- Case $c_1;;\ c_2$: by case distinction:
  - $c_1 = SKIP \Longrightarrow \neg\ \mathit{final}\ (c_1;;\ c_2,\ s)$
  - $c_1 \neq SKIP \Longrightarrow \neg\ \mathit{final}\ (c_1,\ s)$ (by IH)

Can execution stop prematurely?

That is, are there any final configs except $(SKIP, s)$?

Lemma $\mathit{final}\ (c,\ s) \Longrightarrow c = SKIP$

We prove the contrapositive

$c \neq SKIP \Longrightarrow \neg\ \mathit{final}(c, s)$

by induction on c.

- Case $c_1;;\ c_2$: by case distinction:
  - $c_1 = SKIP \Longrightarrow \neg\ \mathit{final}\ (c_1;;\ c_2,\ s)$
  - $c_1 \neq SKIP \Longrightarrow \neg\ \mathit{final}\ (c_1,\ s)$ (by IH)
    $\Longrightarrow \neg\ \mathit{final}\ (c_1;;\ c_2,\ s)$

Can execution stop prematurely?

That is, are there any final configs except $(SKIP, s)$?

Lemma $\mathit{final}\ (c,\ s) \Longrightarrow c = SKIP$

We prove the contrapositive

$$c \neq SKIP \Longrightarrow \neg\ \mathit{final}(c, s)$$

by induction on c.

- Case $c_1;;\ c_2$: by case distinction:
  - $c_1 = SKIP \Longrightarrow \neg\ \mathit{final}\ (c_1;;\ c_2,\ s)$
  - $c_1 \neq SKIP \Longrightarrow \neg\ \mathit{final}\ (c_1,\ s)$ (by IH)
    $\Longrightarrow \neg\ \mathit{final}\ (c_1;;\ c_2,\ s)$
- Remaining cases: trivial or easy

By rule inversion: $(SKIP,\ s) \to ct \Longrightarrow False$

By rule inversion: $(SKIP,\ s) \rightarrow ct \Longrightarrow False$

Together:

Corollary $\mathit{final}\ (c,\ s) = (c = SKIP)$

Infinite executions

$\Rightarrow$ yields final state iff $\rightarrow$ terminates

Infinite executions

$\Rightarrow$ yields final state iff $\rightarrow$ terminates

Lemma $(\exists\, t.\ cs \Rightarrow\ t) = (\exists\, cs'.\ cs \rightarrow\!*\ cs' \wedge \mathit{final}\ cs')$

Infinite executions

$\Rightarrow$ yields final state iff $\rightarrow$ terminates

Lemma $(\exists\, t.\ cs \Rightarrow t) = (\exists\, cs'.\ cs \rightarrow\!* \ cs' \wedge \mathit{final}\ cs')$

Proof: $(\exists\, t.\ cs \Rightarrow t)$

Infinite executions

$\Rightarrow$ yields final state iff $\rightarrow$ terminates

Lemma $(\exists\, t.\ cs \Rightarrow\ t) = (\exists\, cs'.\ cs \rightarrow\!*\ cs' \wedge \mathit{final}\ cs')$

Proof: $(\exists\, t.\ cs \Rightarrow\ t)$

$= \quad (\exists\, t.\ cs \rightarrow\!*\ (\mathit{SKIP},t))$

Infinite executions

$\Rightarrow$ yields final state iff $\rightarrow$ terminates

Lemma $(\exists\, t.\ cs \Rightarrow\ t) = (\exists\, cs'.\ cs \rightarrow\!*\ cs' \wedge \mathit{final}\ cs')$

Proof: $(\exists\, t.\ cs \Rightarrow\ t)$

$= \ (\exists\, t.\ cs \rightarrow\!*\ (\mathit{SKIP},t))$

(by big = small)

Infinite executions

$\Rightarrow$ yields final state iff $\rightarrow$ terminates

Lemma $(\exists\, t.\ cs \Rightarrow t) = (\exists\, cs'.\ cs \rightarrow\!* \ cs' \wedge \mathit{final}\ cs')$

Proof: $(\exists\, t.\ cs \Rightarrow t)$

$= (\exists\, t.\ cs \rightarrow\!* \ (SKIP, t))$

(by big = small)

$= (\exists\, cs'.\ cs \rightarrow\!* \ cs' \wedge \mathit{final}\ cs')$

Infinite executions

$\Rightarrow$ yields final state iff $\rightarrow$ terminates

Lemma $(\exists\, t.\ cs \Rightarrow t) = (\exists\, cs'.\ cs \rightarrow\!\!* \ cs' \wedge \mathit{final}\ cs')$

Proof: $(\exists\, t.\ cs \Rightarrow t)$

$= (\exists\, t.\ cs \rightarrow\!\!* \ (\mathit{SKIP}, t))$

(by big = small)

$= (\exists\, cs'.\ cs \rightarrow\!\!* \ cs' \wedge \mathit{final}\ cs')$

(by final = SKIP)

Infinite executions

$\Rightarrow$ yields final state iff $\rightarrow$ terminates

Lemma $(\exists\, t.\ cs \Rightarrow t) = (\exists\, cs'.\ cs \rightarrow* cs' \wedge \mathit{final}\ cs')$

Proof: $(\exists\, t.\ cs \Rightarrow t)$

$= (\exists\, t.\ cs \rightarrow* (SKIP,t))$

(by big = small)

$= (\exists\, cs'.\ cs \rightarrow* cs' \wedge \mathit{final}\ cs')$

(by final = SKIP)

Equivalent:

$\Rightarrow$ does not yield final state iff $\rightarrow$ does not terminate

May versus Must

$\rightarrow$ is deterministic:

May versus Must

$\rightarrow$ is deterministic:

Lemma $cs \rightarrow cs' \implies cs \rightarrow cs'' \implies cs'' = cs'$

May versus Must

$\rightarrow$ is deterministic:

Lemma $cs \rightarrow cs' \implies cs \rightarrow cs'' \implies cs'' = cs'$
(Proof by rule induction)

May versus Must

$\rightarrow$ is deterministic:

Lemma $cs \rightarrow cs' \implies cs \rightarrow cs'' \implies cs'' = cs'$
(Proof by rule induction)

Therefore: no difference between

may terminate (there is a terminating $\rightarrow$ path)

must terminate (all $\rightarrow$ paths terminate)

May versus Must

$\rightarrow$ is deterministic:

Lemma $cs \rightarrow cs' \implies cs \rightarrow cs'' \implies cs'' = cs'$
(Proof by rule induction)

Therefore: no difference between

may terminate (there is a terminating $\rightarrow$ path)

must terminate (all $\rightarrow$ paths terminate)

Therefore: $\Rightarrow$ correctly reflects termination behaviour.

May versus Must

$\rightarrow$ is deterministic:

Lemma $cs \rightarrow cs' \implies cs \rightarrow cs'' \implies cs'' = cs'$
(Proof by rule induction)

Therefore: no difference between

may terminate (there is a terminating $\rightarrow$ path)

must terminate (all $\rightarrow$ paths terminate)

Therefore: $\Rightarrow$ correctly reflects termination behaviour.

With nondeterminism: may have both $cs \Rightarrow t$ and a nonterminating reduction $cs \rightarrow cs' \rightarrow \ldots$

Chapter 8

Hoare Logic

4 Partial Correctness

5 Total Correctness

4 Partial Correctness

5 Total Correctness

4 Partial Correctness

We have proved functional programs correct (e.g. globbing algorithm)

We have proved functional programs correct (e.g. globbing algorithm)

We have defined two semantics of IMP, and proved equivalence

We have proved functional programs correct (e.g. globbing algorithm)

We have defined two semantics of IMP, and proved equivalence

So how do we prove properties of IMP programs?

An example program:

$$''y'' ::= N\ 0;;\ wsum$$

where

$$
\begin{array}{l}
wsum \equiv \\
WHILE\ Less\ (N\ 0)\ (V\ ''x'') \\
DO\ (''y'' ::= Plus\ (V\ ''y'')\ (V\ ''x'');; \\
\qquad ''x'' ::= Plus\ (V\ ''x'')\ (N\ (-\ 1)))
\end{array}
$$

An example program:

$$''y'' ::= N\ 0;;\ wsum$$

where

$$\begin{array}{l} wsum \equiv \\ WHILE\ Less\ (N\ 0)\ (V\ ''x'') \\ DO\ (''y'' ::= Plus\ (V\ ''y'')\ (V\ ''x'');; \\ \qquad ''x'' ::= Plus\ (V\ ''x'')\ (N\ (-\ 1))) \end{array}$$

At the end of the execution of $''y'' ::= N\ 0;;\ wsum$

An example program:

$''y'' ::= N\ 0;;\ wsum$

where

$$
\begin{array}{l}
wsum \equiv \\
WHILE\ Less\ (N\ 0)\ (V\ ''x'') \\
DO\ (''y'' ::= Plus\ (V\ ''y'')\ (V\ ''x'');; \\
\qquad ''x'' ::= Plus\ (V\ ''x'')\ (N\ (-\ 1)))
\end{array}
$$

At the end of the execution of $''y'' ::= N\ 0;;\ wsum$ variable $''y''$ should contain the sum $1 + \ldots + i$

An example program:

$''y'' ::= N\ 0;;\ wsum$

where

$$
\begin{array}{l}
wsum \equiv \\
WHILE\ Less\ (N\ 0)\ (V\ ''x'') \\
DO\ (''y'' ::= Plus\ (V\ ''y'')\ (V\ ''x'');; \\
\qquad ''x'' ::= Plus\ (V\ ''x'')\ (N\ (-\ 1)))
\end{array}
$$

At the end of the execution of $''y'' ::= N\ 0;;\ wsum$ variable $''y''$ should contain the sum $1 + \ldots + i$ where i is the initial value of $''x''$.

An example program:

$$''y'' ::= N\ 0;;\ wsum$$

where

$$\begin{array}{l} wsum \equiv \\ WHILE\ Less\ (N\ 0)\ (V\ ''x'') \\ DO\ (''y'' ::= Plus\ (V\ ''y'')\ (V\ ''x'');; \\ \qquad ''x'' ::= Plus\ (V\ ''x'')\ (N\ (-\ 1))) \end{array}$$

At the end of the execution of $''y'' ::= N\ 0;;\ wsum$ variable $''y''$ should contain the sum $1 + \ldots + i$ where i is the initial value of $''x''$.

$$sum\ i = (\textit{if}\ i \leq 0\ \textit{then}\ 0\ \textit{else}\ sum\ (i - 1) + i)$$

A proof via operational semantics

Theorem:

$(''y'' ::= N\ 0;;\ wsum,\ s) \Rightarrow t \Longrightarrow$
$t\ ''y'' = sum\ (s\ ''x'')$

A proof via operational semantics

Theorem:

$$(''y'' ::= N\,0;;\ wsum,\ s) \Rightarrow t \Longrightarrow$$
$$t\ ''y'' = sum\ (s\ ''x'')$$

Required Lemma:

$$(wsum,\ s) \Rightarrow t \Longrightarrow$$
$$t\ ''y'' = s\ ''y'' + sum\ (s\ ''x'')$$

A proof via operational semantics

Theorem:

$$(''y'' ::= N\,0;;\ wsum,\ s) \Rightarrow t \Longrightarrow$$
$$t\ ''y'' = sum\ (s\ ''x'')$$

Required Lemma:

$$(wsum,\ s) \Rightarrow t \Longrightarrow$$
$$t\ ''y'' = s\ ''y'' + sum\ (s\ ''x'')$$

Proved by rule induction.

Hoare Logic provides a *structured* approach for reasoning about properties of states during program execution:

Hoare Logic provides a *structured* approach for reasoning about properties of states during program execution:

- Rules of Hoare Logic (almost) syntax directed

Hoare Logic provides a *structured* approach for reasoning about properties of states during program execution:

- Rules of Hoare Logic (almost) syntax directed
- Automates reasoning about program execution

Hoare Logic provides a *structured* approach for reasoning about properties of states during program execution:

- Rules of Hoare Logic (almost) syntax directed
- Automates reasoning about program execution
- No explicit induction

Hoare Logic provides a *structured* approach for reasoning about properties of states during program execution:

- Rules of Hoare Logic (almost) syntax directed
- Automates reasoning about program execution
- No explicit induction

But no free lunch:

Hoare Logic provides a *structured* approach for reasoning about properties of states during program execution:

- Rules of Hoare Logic (almost) syntax directed
- Automates reasoning about program execution
- No explicit induction

But no free lunch:

- Must prove implications between predicates on states

Hoare Logic provides a *structured* approach for reasoning about properties of states during program execution:

- Rules of Hoare Logic (almost) syntax directed
- Automates reasoning about program execution
- No explicit induction

But no free lunch:

- Must prove implications between predicates on states
- Needs invariants.

4 Partial Correctness

This is the standard approach.

This is the standard approach. Originally with pen and paper.

This is the standard approach. Originally with pen and paper.

Formulas are syntactic objects.

This is the standard approach. Originally with pen and paper.

Formulas are syntactic objects.

Everything is very concrete and simple.

This is the standard approach. Originally with pen and paper.

Formulas are syntactic objects.

Everything is very concrete and simple.

But complex to formalize.

This is the standard approach. Originally with pen and paper.

Formulas are syntactic objects.

Everything is very concrete and simple.

But complex to formalize.

Hence we soon move to a semantic view of formulas.

This is the standard approach. Originally with pen and paper.

Formulas are syntactic objects.

Everything is very concrete and simple.

But complex to formalize.

Hence we soon move to a semantic view of formulas.

Reason for introduction of syntactic approach: didactic

This is the standard approach. Originally with pen and paper.

Formulas are syntactic objects.

Everything is very concrete and simple.

But complex to formalize.

Hence we soon move to a semantic view of formulas.

Reason for introduction of syntactic approach: didactic

Hoare Logic reasons about *Hoare triples* $\{P\}\ c\ \{Q\}$ where

Hoare Logic reasons about *Hoare triples* $\{P\}\ c\ \{Q\}$ where

- P and Q are *syntactic formulas* involving program variables

Hoare Logic reasons about *Hoare triples* $\{P\}\ c\ \{Q\}$ where

- P and Q are *syntactic formulas* involving program variables
- P is the *precondition*,

Hoare Logic reasons about *Hoare triples* $\{P\}\ c\ \{Q\}$ where

- P and Q are *syntactic formulas* involving program variables
- P is the *precondition*, Q is the *postcondition*

Hoare Logic reasons about *Hoare triples* $\{P\}\ c\ \{Q\}$ where

- P and Q are *syntactic formulas* involving program variables
- P is the *precondition*, Q is the *postcondition*
- $\{P\}\ c\ \{Q\}$ means that

Hoare Logic reasons about *Hoare triples* $\{P\}\ c\ \{Q\}$ where

- P and Q are *syntactic formulas* involving program variables
- P is the *precondition*, Q is the *postcondition*
- $\{P\}\ c\ \{Q\}$ means that if P is true at the start of the execution,

Hoare Logic reasons about *Hoare triples* $\{P\}\ c\ \{Q\}$ where

- P and Q are *syntactic formulas* involving program variables
- P is the *precondition*, Q is the *postcondition*
- $\{P\}\ c\ \{Q\}$ means that if P is true at the start of the execution, Q is true at the end of the execution

Hoare Logic reasons about *Hoare triples* $\{P\}\ c\ \{Q\}$ where

- P and Q are *syntactic formulas* involving program variables
- P is the *precondition*, Q is the *postcondition*
- $\{P\}\ c\ \{Q\}$ means that if P is true at the start of the execution, Q is true at the end of the execution — if the execution terminates!

Hoare Logic reasons about *Hoare triples* $\{P\}\ c\ \{Q\}$ where

- P and Q are *syntactic formulas* involving program variables
- P is the *precondition*, Q is the *postcondition*
- $\{P\}\ c\ \{Q\}$ means that
  if P is true at the start of the execution,
  Q is true at the end of the execution
  — if the execution terminates! (*partial correctness*)

Hoare Logic reasons about *Hoare triples* $\{P\}\ c\ \{Q\}$ where

- P and Q are *syntactic formulas* involving program variables
- P is the *precondition*, Q is the *postcondition*
- $\{P\}\ c\ \{Q\}$ means that if P is true at the start of the execution, Q is true at the end of the execution — if the execution terminates! (*partial correctness*)

Informal example:

$$\{x = 41\}\ x := x + 1\ \{x = 42\}$$

Hoare Logic reasons about *Hoare triples* $\{P\}\ c\ \{Q\}$ where

- P and Q are *syntactic formulas* involving program variables
- P is the *precondition*, Q is the *postcondition*
- $\{P\}\ c\ \{Q\}$ means that if P is true at the start of the execution, Q is true at the end of the execution — if the execution terminates! (*partial correctness*)

Informal example:

$$\{x = 41\}\ x := x + 1\ \{x = 42\}$$

Terminology: P and Q are called *assertions*.

Examples

$$\{x = 5\} \qquad ? \qquad \{x = 10\}$$

Examples

$$\{x = 5\} \qquad ? \qquad \{x = 10\}$$

$$\{\mathit{True}\} \qquad ? \qquad \{x = 10\}$$

Examples

$$\{x = 5\} \quad ? \quad \{x = 10\}$$

$$\{True\} \quad ? \quad \{x = 10\}$$

$$\{x = y\} \quad ? \quad \{x \neq y\}$$

Examples

$$\{x = 5\} \quad ? \quad \{x = 10\}$$

$$\{True\} \quad ? \quad \{x = 10\}$$

$$\{x = y\} \quad ? \quad \{x \neq y\}$$

Boundary cases:

$$\{True\} \quad ? \quad \{True\}$$

Examples

$$\{x = 5\} \quad ? \quad \{x = 10\}$$

$$\{True\} \quad ? \quad \{x = 10\}$$

$$\{x = y\} \quad ? \quad \{x \neq y\}$$

Boundary cases:

$$\{True\} \quad ? \quad \{True\}$$

$$\{True\} \quad ? \quad \{False\}$$

Examples

$$\{x = 5\} \quad ? \quad \{x = 10\}$$

$$\{True\} \quad ? \quad \{x = 10\}$$

$$\{x = y\} \quad ? \quad \{x \neq y\}$$

Boundary cases:

$$\{True\} \quad ? \quad \{True\}$$

$$\{True\} \quad ? \quad \{False\}$$

$$\{False\} \quad ? \quad \{Q\}$$

The rules of Hoare Logic

The rules of Hoare Logic

$$(\textsc{Skip}) \ \frac{-}{\{P\}\ SKIP\ \{P\}}$$

The rules of Hoare Logic

$$(\textsc{Skip})\ \frac{-}{\{P\}\ SKIP\ \{P\}}$$

$$(\textsc{Assign})\ \frac{-}{\{Q[a/x]\}\ x := a\ \{Q\}}$$

The rules of Hoare Logic

$$(\textsc{Skip})\ \frac{-}{\{P\}\ SKIP\ \{P\}}$$

$$(\textsc{Assign})\ \frac{-}{\{Q[a/x]\}\ x := a\ \{Q\}}$$

Notation: $Q[a/x]$ means "Q with a substituted for x".

The rules of Hoare Logic

$$(\textsc{Skip})\ \frac{-}{\{P\}\ SKIP\ \{P\}}$$

$$(\textsc{Assign})\ \frac{-}{\{Q[a/x]\}\ x := a\ \{Q\}}$$

Notation: $Q[a/x]$ means "Q with a substituted for x".

Examples: $\{\qquad\}\ x := 5 \qquad\qquad \{x = 5\}$

The rules of Hoare Logic

$$(\textsc{Skip})\ \frac{-}{\{P\}\ SKIP\ \{P\}}$$

$$(\textsc{Assign})\ \frac{-}{\{Q[a/x]\}\ x := a\ \{Q\}}$$

Notation: $Q[a/x]$ means "Q with a substituted for x".

Examples: $\{\qquad\}\ x := 5 \qquad \{x = 5\}$
$\{\qquad\}\ x := x{+}5 \qquad \{x = 5\}$

The rules of Hoare Logic

$$(\textsc{Skip})\ \frac{-}{\{P\}\ SKIP\ \{P\}}$$

$$(\textsc{Assign})\ \frac{-}{\{Q[a/x]\}\ x := a\ \{Q\}}$$

Notation: $Q[a/x]$ means "Q with a substituted for x".

Examples:

$$\begin{array}{lll} \{\qquad\} & x := 5 & \{x = 5\} \\ \{\qquad\} & x := x{+}5 & \{x = 5\} \\ \{\qquad\} & x := 2{*}(x{+}5) & \{x > 20\} \end{array}$$

The rules of Hoare Logic

$$(\textsc{Skip})\ \frac{-}{\{P\}\ SKIP\ \{P\}}$$

$$(\textsc{Assign})\ \frac{-}{\{Q[a/x]\}\ x := a\ \{Q\}}$$

Notation: $Q[a/x]$ means "Q with a substituted for x".

Examples:

$$\begin{array}{llll} \{ & \} & x := 5 & \{x = 5\} \\ \{ & \} & x := x{+}5 & \{x = 5\} \\ \{ & \} & x := 2*(x{+}5) & \{x > 20\} \end{array}$$

Alternative explanation of assignment rule:

$$\{Q[a]\}\ x := a\ \{Q[x]\}$$

The assignment rule allows us
to compute the precondition from the postcondition.

The assignment rule allows us
to compute the precondition from the postcondition.

There is a version to compute the postcondition from the precondition, but it is more complicated.

The assignment rule allows us
to compute the precondition from the postcondition.

There is a version to compute the postcondition from the precondition, but it is more complicated. (Exercise!)

More rules of Hoare Logic

More rules of Hoare Logic

$$(\textsc{Seq})\ \frac{}{\{P_1\}\ c_1;c_2\ \{P_3\}}$$

More rules of Hoare Logic

$$(\textsc{Seq})\ \frac{\{P_1\}\ c_1\ \{P_2\} \qquad \{P_2\}\ c_2\ \{P_3\}}{\{P_1\}\ c_1; c_2\ \{P_3\}}$$

More rules of Hoare Logic

$$(\textsc{Seq})\ \frac{\{P_1\}\ c_1\ \{P_2\} \qquad \{P_2\}\ c_2\ \{P_3\}}{\{P_1\}\ c_1;c_2\ \{P_3\}}$$

$$(\textsc{If})\ \frac{}{\{P\}\ IF\ b\ THEN\ c_1\ ELSE\ c_2\ \{Q\}}$$

More rules of Hoare Logic

$$(\textsc{Seq})\ \frac{\{P_1\}\ c_1\ \{P_2\} \qquad \{P_2\}\ c_2\ \{P_3\}}{\{P_1\}\ c_1;c_2\ \{P_3\}}$$

$$(\textsc{If})\ \frac{\{P \wedge b\}\ c_1\ \{Q\} \qquad \{P \wedge \neg\ b\}\ c_2\ \{Q\}}{\{P\}\ IF\ b\ THEN\ c_1\ ELSE\ c_2\ \{Q\}}$$

More rules of Hoare Logic

$$(\textsc{Seq})\ \frac{\{P_1\}\ c_1\ \{P_2\} \qquad \{P_2\}\ c_2\ \{P_3\}}{\{P_1\}\ c_1;c_2\ \{P_3\}}$$

$$(\textsc{If})\ \frac{\{P \land b\}\ c_1\ \{Q\} \qquad \{P \land \neg\ b\}\ c_2\ \{Q\}}{\{P\}\ \mathit{IF}\ b\ \mathit{THEN}\ c_1\ \mathit{ELSE}\ c_2\ \{Q\}}$$

$$(\textsc{While})\ \frac{}{\{P\}\ \mathit{WHILE}\ b\ \mathit{DO}\ c\ \{P \land \neg\ b\}}$$

More rules of Hoare Logic

$$(\textsc{Seq})\ \frac{\{P_1\}\ c_1\ \{P_2\} \qquad \{P_2\}\ c_2\ \{P_3\}}{\{P_1\}\ c_1;c_2\ \{P_3\}}$$

$$(\textsc{If})\ \frac{\{P \wedge b\}\ c_1\ \{Q\} \qquad \{P \wedge \neg\ b\}\ c_2\ \{Q\}}{\{P\}\ \mathit{IF}\ b\ \mathit{THEN}\ c_1\ \mathit{ELSE}\ c_2\ \{Q\}}$$

$$(\textsc{While})\ \frac{\{P \wedge b\}\ c\ \{P\}}{\{P\}\ \mathit{WHILE}\ b\ \mathit{DO}\ c\ \{P \wedge \neg\ b\}}$$

More rules of Hoare Logic

$$(\textsc{Seq})\ \frac{\{P_1\}\ c_1\ \{P_2\} \qquad \{P_2\}\ c_2\ \{P_3\}}{\{P_1\}\ c_1;c_2\ \{P_3\}}$$

$$(\textsc{If})\ \frac{\{P \wedge b\}\ c_1\ \{Q\} \qquad \{P \wedge \neg\, b\}\ c_2\ \{Q\}}{\{P\}\ \mathit{IF}\ b\ \mathit{THEN}\ c_1\ \mathit{ELSE}\ c_2\ \{Q\}}$$

$$(\textsc{While})\ \frac{\{P \wedge b\}\ c\ \{P\}}{\{P\}\ \mathit{WHILE}\ b\ \mathit{DO}\ c\ \{P \wedge \neg\, b\}}$$

In the While-rule, P is called an *invariant* because it is preserved across executions of the loop body.

The *consequence* rule

The *consequence* rule

So far, the rules were syntax-directed.

The *consequence* rule

So far, the rules were syntax-directed. Now we add

$$(\textsc{Cons})\ \frac{P' \longrightarrow P \qquad \{P\}\ c\ \{Q\} \qquad Q \longrightarrow Q'}{\{P'\}\ c\ \{Q'\}}$$

The *consequence* rule

So far, the rules were syntax-directed. Now we add

$$(\textsc{Cons})\ \frac{P' \longrightarrow P \qquad \{P\}\ c\ \{Q\} \qquad Q \longrightarrow Q'}{\{P'\}\ c\ \{Q'\}}$$

Preconditions can be strengthened,
postconditions can be weakened.

Derived While Rule

Problem with While-rule:
special form of pre and postcondition.

Derived While Rule

Problem with While-rule:
special form of pre and postcondition.
Better: combine with consequence rule.

Derived While Rule

Problem with While-rule:
special form of pre and postcondition.
Better: combine with consequence rule.

$$(\textsc{While'})\ \frac{}{\{P\}\ \mathit{WHILE}\ b\ \mathit{DO}\ c\ \{Q\}}$$

Derived While Rule

Problem with While-rule:
special form of pre and postcondition.
Better: combine with consequence rule.

$$(\textsc{While'})\ \frac{\{R\}\ c\ \{P\} \qquad P \wedge b \longrightarrow R \qquad P \wedge \neg\, b \longrightarrow Q}{\{P\}\ \mathit{WHILE}\ b\ \mathit{DO}\ c\ \{Q\}}$$

Example

$$\{x = i\}$$

$$y := 0;$$
$$\mathit{WHILE}\ 0 < x\ \mathit{DO}\ (y := y{+}x;\ x := x{-}1)$$

$$\{y = sum\ i\}$$

Example proof exhibits key properties of Hoare logic:

Example proof exhibits key properties of Hoare logic:

- Choice of rules is syntax-directed and hence automatic.

Example proof exhibits key properties of Hoare logic:

- Choice of rules is syntax-directed and hence automatic.
- Proof of ";" proceeds from right to left.

Example proof exhibits key properties of Hoare logic:

- Choice of rules is syntax-directed and hence automatic.
- Proof of ";" proceeds from right to left.
- Proofs require only invariants and arithmetic reasoning.

4 Partial Correctness

Introduction

Syntactic Approach

Formalizing Hoare Logic

Assertions

Assertions are predicates on states

Assertions

Assertions are predicates on states

$$assn = state \Rightarrow bool$$

Assertions

Assertions are predicates on states

$$assn = state \Rightarrow bool$$

Alternative view: *sets of states*

Assertions

Assertions are predicates on states

$$assn = state \Rightarrow bool$$

Alternative view: *sets of states*

Hoare Triples

$$\models \{P\}\ c\ \{Q\}$$

$$\longleftrightarrow$$

$$\forall\, s\ t.\ P\ s \wedge (c,\ s) \Rightarrow t \longrightarrow Q\ t$$

Hoare Triples

$$\models \{P\}\ c\ \{Q\}$$

$$\longleftrightarrow$$

$$\forall\, s\ t.\ P\ s \wedge (c,\ s) \Rightarrow t \longrightarrow Q\ t$$

"$\{P\}\ c\ \{Q\}$ is valid"

Proving Hoare Triples

Straightforward for most commands

Proving Hoare Triples

Straightforward for most commands

$$\models \{\lambda s.\ Q\ (s(x := aval\ a\ s))\}\ x ::= a\ \{Q\}$$

Proving Hoare Triples

Straightforward for most commands

$$\models \{\lambda s.\ Q\ (s(x := aval\ a\ s))\}\ x ::= a\ \{Q\}$$

Using semantic substitution.

Proving Hoare Triples

Straightforward for most commands

$$\models \{\lambda s.\ Q\ (s(x := aval\ a\ s))\}\ x ::= a\ \{Q\}$$

Using semantic substitution.

While-rule requires induction

Hoare.thy

Verification Condition Generation

Recall: Applying the Hoare rules is automatic

Verification Condition Generation

Recall: Applying the Hoare rules is automatic

Except for coming up with invariants.

Verification Condition Generation

Recall: Applying the Hoare rules is automatic

Except for coming up with invariants.

And discharging (some) VCs.

Verification Condition Generation

Recall: Applying the Hoare rules is automatic

Except for coming up with invariants.

And discharging (some) VCs.

Annotate invariants to program

Verification Condition Generation

Recall: Applying the Hoare rules is automatic

Except for coming up with invariants.

And discharging (some) VCs.

Annotate invariants to program

Then generate VCs automatically

Verification Condition Generation

Recall: Applying the Hoare rules is automatic

Except for coming up with invariants.

And discharging (some) VCs.

Annotate invariants to program

Then generate VCs automatically

Use auxiliary lemmas to discharge VCs

VCG.thy

4 Partial Correctness

5 Total Correctness

- Partial Correctness:
  if command terminates, postcondition holds

- Partial Correctness:
  if command terminates, postcondition holds
- Total Correctness:
  command terminates *and* postcondition holds

- Partial Correctness:
  if command terminates, postcondition holds
- Total Correctness:
  command terminates *and* postcondition holds

Total Correctness = Partial Correctness + Termination

- Partial Correctness:
  if command terminates, postcondition holds
- Total Correctness:
  command terminates *and* postcondition holds

Total Correctness = Partial Correctness + Termination

Formally:

$$(\models_t \{P\}\ c\ \{Q\}) =$$
$$(\forall s.\ P\ s \longrightarrow (\exists t.\ (c,\ s) \Rightarrow t \wedge\ Q\ t))$$

- Partial Correctness:
  if command terminates, postcondition holds
- Total Correctness:
  command terminates *and* postcondition holds

Total Correctness = Partial Correctness + Termination

Formally:

$$(\models_t \{P\}\ c\ \{Q\}) =$$
$$(\forall s.\ P\ s \longrightarrow (\exists t.\ (c,\ s) \Rightarrow t \land\ Q\ t))$$

Assumes that semantics is deterministic!

- Partial Correctness:
  if command terminates, postcondition holds
- Total Correctness:
  command terminates *and* postcondition holds

Total Correctness $=$ Partial Correctness $+$ Termination

Formally:

$$(\models_t \{P\}\ c\ \{Q\}) =$$
$$(\forall s.\ P\ s \longrightarrow (\exists t.\ (c,\ s) \Rightarrow t \wedge\ Q\ t))$$

Assumes that semantics is deterministic!

$\models_t$: Proving Total Correctness

$\models_t$: Proving Total Correctness

Only need to change the $WHILE$ rule.

$\models_t$: Proving Total Correctness

Only need to change the $WHILE$ rule. In each iteration, the state must decrease wrt. a well-founded relation

$\models_t$: Proving Total Correctness

Only need to change the $WHILE$ rule. In each iteration, the state must decrease wrt. a well-founded relation

$$\frac{wf\ R \qquad \bigwedge s_0.\ \models_t \{\lambda s.\ P\ s \wedge bval\ b\ s \wedge s = s_0\}\ c\ \{\lambda s.\ P\ s \wedge (s,\ s_0) \in R\}}{\models_t \{P\}\ WHILE\ b\ DO\ c\ \{\lambda s.\ P\ s \wedge \neg\ bval\ b\ s\}}$$

Well-Founded Relations

A relation $<$ is well-founded, if there is no infinite down-chain.

Well-Founded Relations

A relation $<$ is well-founded, if there is no infinite down-chain.

$$\ldots < s_3 < s_2 < s_1 < s_0$$

Well-Founded Relations

A relation $<$ is well-founded, if there is no infinite down-chain.

$$\ldots < s_3 < s_2 < s_1 < s_0$$

Induction principle: Show $P\ a$, assuming P holds for smaller states

Well-Founded Relations

A relation $<$ is well-founded, if there is no infinite down-chain.

$$\ldots < s_3 < s_2 < s_1 < s_0$$

Induction principle: Show $P\ a$, assuming P holds for smaller states

$$\frac{wf\ r \qquad \bigwedge x.\ \dfrac{\forall\, y.\ (y,\, x) \in r \longrightarrow P\ y}{P\ x}}{P\ a}$$

Well-Founded Relations

A relation $<$ is well-founded, if there is no infinite down-chain.

$$\ldots \; < \; s_3 \; < \; s_2 \; < \; s_1 \; < \; s_0$$

Induction principle: Show $P\ a$, assuming P holds for smaller states

$$\frac{wf\ r \qquad \bigwedge x.\ \dfrac{\forall\, y.\ (y,\ x) \in r \longrightarrow P\ y}{P\ x}}{P\ a}$$

Definition of wf: Induction principle holds!

$wf\ r = (\forall\, P.\ (\forall\, x.\ (\forall\, y.\ (y,\ x) \in r \longrightarrow P\ y) \longrightarrow P\ x)$
$\longrightarrow (\forall\, x.\ P\ x))$

Soundness

Proof by well-founded induction

IMPArrayHoareTotal.thy

While-Rule for Total Correctness

Measure Functions

Function $m :: state \Rightarrow nat$. Must decrease in each loop iteration.

Measure Functions

Function $m :: state \Rightarrow nat$. Must decrease in each loop iteration.

$$measure\ m = \{(x,\ y).\ m\ x < m\ y\}$$

Measure Functions

Function $m :: state \Rightarrow nat$. Must decrease in each loop iteration.

$measure\ m = \{(x,\ y).\ m\ x < m\ y\}$

Measures always well-founded!

$wf\ (measure\ f)$

IMPAHP_Examples.thy

Total Correctness Examples